下载APP

18 | 场景联动：智能电灯如何感知光线？（上）

2020-12-21 郭朝斌

物联网开发实战 进入课程 >

讲述：郭朝斌
时长 13:21 大小 12.24M

你好，我是郭朝斌。

在上一讲，我们打造了自己的联网智能电灯，你可以通过手机小程序来控制它的打开和关闭，也就是实现远程控制。

其实，我们还可以进一步提高体验，让智能电灯可以基于环境的明暗来自动地打开和关闭。要做到这一点并不难，可以分为两个阶段，第一阶段是打造传感器设备来感知光照的强弱，判断出环境的明暗状态，第二阶段是创建一个场景联动，根据传感器的数值来控制智能电灯的状态。

这一讲，我先带你一步一步地实现第一阶段的工作。

第一步：通信技术

首先，我们为光照传感器设备选择通信技术。

因为光照传感器设备的部署位置比较灵活，不太可能像智能电灯一样连接房间里的电源线，所以我们要用一种比 Wi-Fi 功耗更低的通信技术。这样的话，就算使用电池供电，也可以长时间（一年以上）持续工作。

经过对比，我建议选择 BLE 低功耗蓝牙技术（关于通信技术的选择策略，你可以参考第 2 讲）。随着智能手机的发展，蓝牙早已成为手机标配的通信技术，蓝牙芯片和协议栈的成熟度非常高，而且在设备的供应链方面，蓝牙芯片可以选择的供应商也非常多。

不过在正式开发之前，我还得为你补充说明一些 BLE 的相关知识。

BLE 设备可以在 4 种模式下工作：

1. **广播模式**（Broadcaster），这里特指单纯的广播模式。这种模式下设备不可以被连接，只能够以一定的时间间隔把数据广播出来，供其他设备使用，比如手机扫描处理。蓝牙 Beacon 设备就是工作在这种模式。
2. **从机模式**（Peripheral），这种模式下设备仍然可以广播数据，同时也可以被连接。建立连接后，双方可以进行双向通信。比如你用手机连接一个具有蓝牙功能的体温计，这时体温计就是从机（Peripheral）。
3. **主机模式**（Central），这种模式下设备不进行广播，但是可以扫描周围的蓝牙广播包，发现其他设备，然后主动对这些设备发起连接。还是刚才那个例子，主动连接蓝牙体温计的手机就是主机（Central）角色。
4. **观察者模式**（Observer），这种模式下设备像主机模式一样，也不进行广播，而是扫描周围的蓝牙广播包，但是不同的地方是，它不会与从机设备建立连接。一般收集蓝牙设备广播包的网关就是在这种模式下工作的，它会将收集的广播数据通过网线、Wi-Fi 或者 4G 等蜂窝网络上传到云平台。

在这一讲中，我们打造的光照传感器只需要提供光照强度数据就行了，并不需要进行双向通信，所以我们可以定义设备在广播模式下工作。

第二步：选择开发板

那么光照传感器设备要选择什么开发板呢？

我们在上一讲打造的联网智能电灯中使用的 NodeMCU 是基于 **ESP8266 芯片**的，相信你也注意到了，这款芯片并不支持低功耗蓝牙。

好在市场上还有一款基于 **ESP32 芯片**的 NodeMCU 开发板。ESP32是乐鑫科技出品的另一款性能优良且满足低功耗的物联网芯片，它同时支持 Wi-Fi 和低功率蓝牙通信技术，还有丰富的 ADC 接口。

更重要的是，MicroPython 也支持 ESP32 芯片，这样我们就可以继续使用 Python 语言来开发了。

第三步：准备 MicroPython 环境

接下来，我们就在 NodeMCU（ESP32）上安装 MicroPython 固件，准备 Python 程序的运行环境。

MicroPython 官网已经为我们准备了编译好的固件文件，这省掉了我们在电脑上进行交叉编译的工作。你可以从这个链接 中选择“Firmware with ESP-IDF v3.x”下面的“GENERIC”类别，直接下载最新版本的固件文件到电脑中。

然后，我们使用一根 USB 数据线，将 NodeMCU 开发板和电脑连接起来。USB 数据线仍然选择一头是 USB-A 接口、另一头是 Micro-USB 接口，并且支持数据传输的完整线缆。具体细节，你可以再回顾第 16 讲中的相关内容。

我们使用 esptool 工具把这个固件烧录到 NodeMCU 开发板上。先在电脑终端上输入下面的命令，清空一下 NodeMCU 的 Flash 存储芯片。

```
1 esptool.py --chip esp32 --port /dev/cu.usbserial-0001 erase_flash
```

复制代码

你可以从命令里看到，和之前智能电灯用的命令相比，这里增加了芯片信息“esp32”。另外，“--port”后面的串口设备名称，需要你替换为自己电脑上对应的名称。

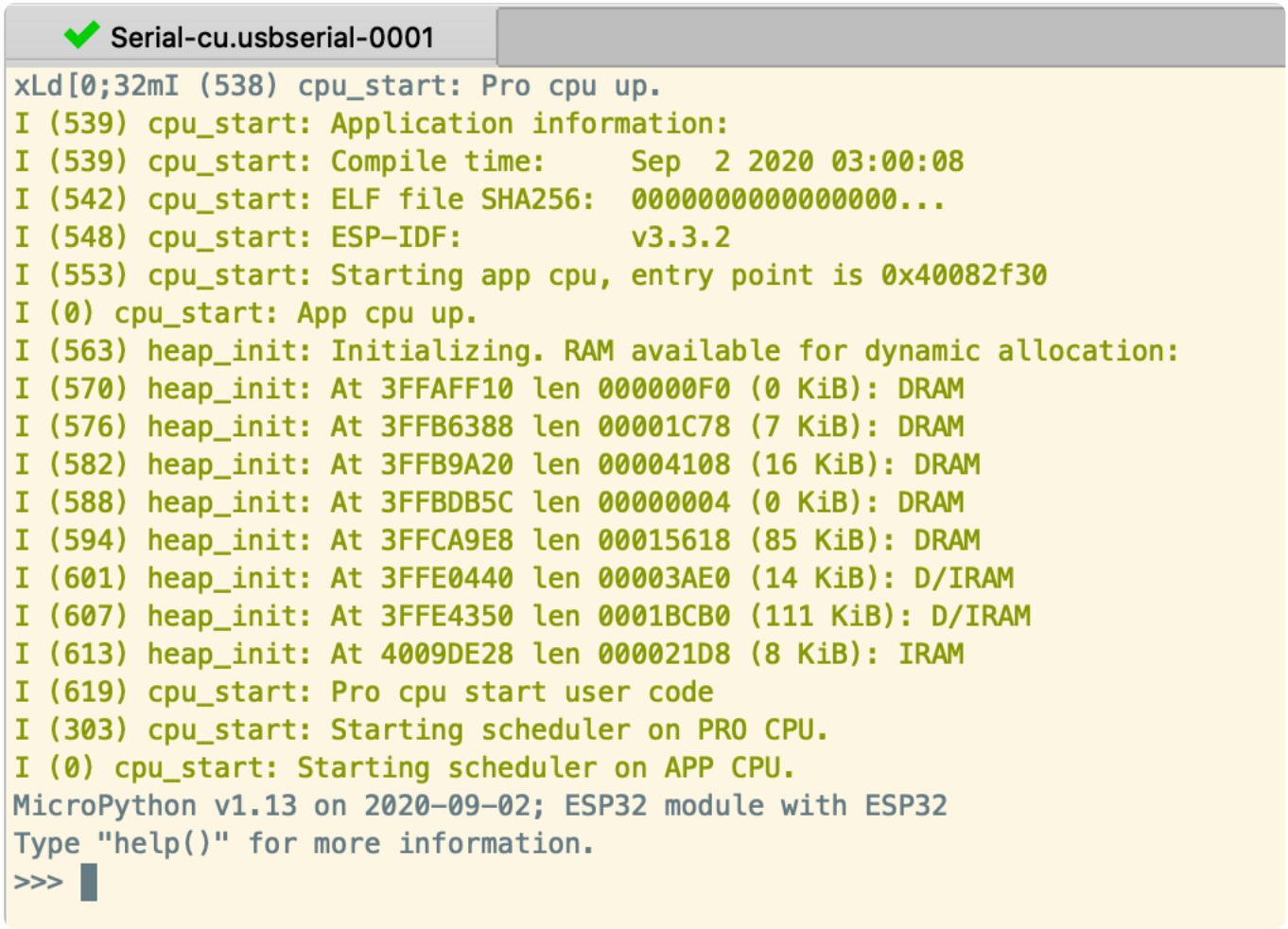
成功擦除 Flash 之后，就执行下面的命令，将固件写入 Flash 芯片。

```
1 esptool.py --chip esp32 --port /dev/cu.usbserial-0001 --baud 460800 write_flas
```

复制代码

这时，我们使用电脑上的终端模拟器软件，比如 SecureCRT，通过串口协议连接上开发板，注意波特率（Baud rate）设置为 115200。

然后你应该就能看到下图所示的内容，并且可以进行交互。



第四步：搭建光照传感器硬件电路

现在，我们开始基于 NodeMCU 搭建光照传感器的硬件电路。

首先，我们要准备好实验的材料：

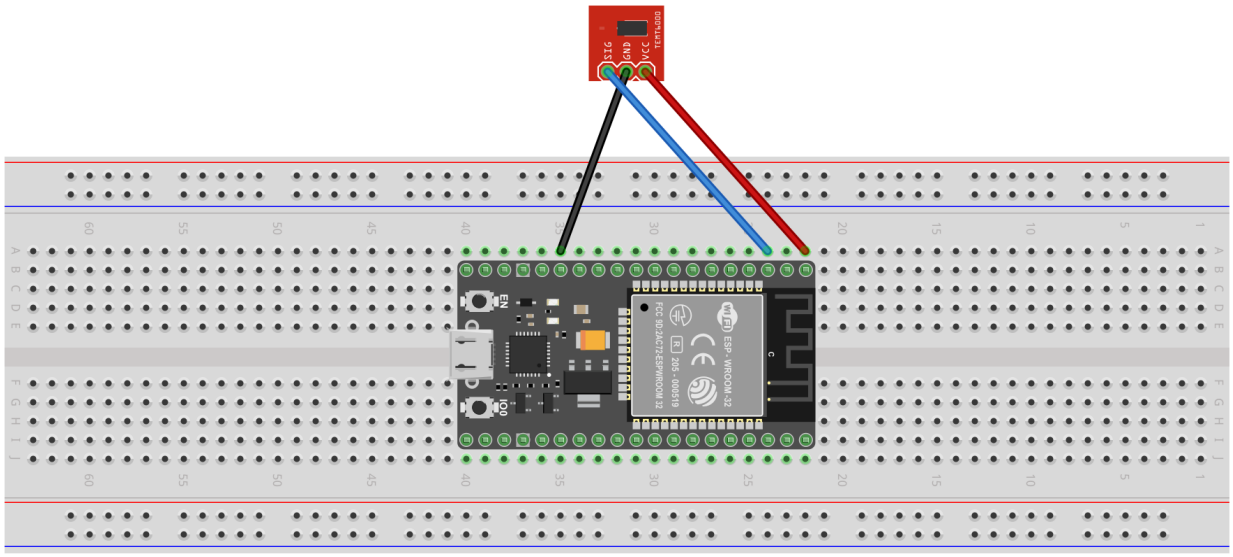
1. NodeMCU（ESP32）开发板一个。注意区分芯片的具体型号。

2. 光照传感器模块一个

3. 杜邦线 / 跳线若干个

4. 面包板一个

然后，你可以按照我画的连线图来搭建出自己的电路。跟联网智能电灯的电路比起来，这个还是非常简单的。



这里说明一下，在我的电路图中，光照传感器模块从左到右，管脚分别是光强度模拟信号输出管脚、电源地 GND 和电源正 VCC 管脚。你需要根据自己的传感器模块调整具体的连线。

我选择的是基于 PT550 环保型光敏二极管的光照传感器元器件，它的灵敏度更高，测量范围是 0Lux ~ 6000Lux。

Lux（勒克斯）是光照强度的单位，它和另一个概念 Lumens（流明）是不同的。Lumens 是指一个光源（比如电灯、投影仪）发出的光能力的总量，而 Lux 是指空间内一个位置接收到的光照的强度。

这个元器件通过信号管脚输出模拟量，我们读取 NodeMCU ESP32 的 ADC 模数转换器（ADC0，对应 GPIO36）的数值，就可以得到光照强度。这个数值越大，表示光照强度越大。

因为 ADC 支持的最大位数是 12bit，所以这个数值范围是 0~4095 之间。这里我们粗略地按照线性关系做一个转换。具体计算过程，你可以参考下面的代码：

复制代码

```
from machine import ADC
from machine import Pin

class LightSensor():

    def __init__(self, pin):
        self.light = ADC(Pin(pin))

    def value(self):
        value = self.light.read()
        print("Light ADC value:",value)
        return int(value/4095*6000)
```

第五步：编写蓝牙程序

NodeMCU ESP32 的固件已经集成了 BLE 的功能，我们可以直接在这个基础上进行软件的开发。这里我们需要给广播包数据定义一定的格式，让其他设备可以顺利地解析使用扫描到的数据。

那么怎么定义蓝牙广播包的格式呢？我们可以使用小米制定的MiBeacon蓝牙协议。

MiBeacon 蓝牙协议的广播包格式是基于 BLE 的 GAP（Generic Access Profile）制定的。GAP 控制了蓝牙的广播和连接，也就是控制了设备如何被发现，以及如何交互。

具体来说，GAP 定义了两种方式来让设备广播数据：

一个是广播数据（Advertising Data payload），这个是必须的，数据长度是 31 个字节；

另一个是扫描回复数据（Scan Response payload），它基于蓝牙主机设备（比如手机）发出的扫描请求（Scan Request）来回复一些额外的信息。数据长度和广播数据一样。

（注意，蓝牙 5.0 中有扩展的广播数据，数据长度等特性与此不同，但这里不涉及，所以不再介绍。）

所以，只要含有以下指定信息的广播报文，就可以认为是符合 MiBeacon 蓝牙协议的。

1. Advertising Data 中 Service Data (0x16) 含有 Mi Service UUID 的广播包，UUID 是 0xFE95。
2. Scan Response 中 Manufacturer Specific Data (0xFF) 含有小米公司识别码的广播包，识别码 ID 是 0x038F。

其中，无论是在 Advertising Data 中，还是 Scan Response 中，均采用统一格式定义。

具体的广播报文格式定义，你可以参考下面的表格。

名称	长度（byte）	是否必须	描述
Frame Control	2	必须	控制位
Product ID	2	必须	产品ID需要向小米申请
Frame Counter	1	必须	序号，用于去重，不同的事件或属性上报需要不同的Frame Counter
MAC Address	6	基于Frame Control	设备Mac地址
Capability	1	基于Frame Control	设备能力
I/O capability	2	基于Capability	I/O能力，目前只有高安全级BLE接入才会用到此字段
Object	若干最大10字节	基于Frame Control	触发事件或广播属性
Random Number	3	基于Frame Control	如果加密则为必选字段，与Frame Counter合并成为4字节Counter，用于防重放
Message Integrity Check	4	基于Frame Control	如果加密则为必选字段

因为我们要为光照传感器增加广播光照强度数据的能力，所以主要关注Object 的定义。Object 分为属性和事件两种，具体定义了设备数据的含义，比如体温计的温度、土壤的湿度等，数据格式如下表所示：

字段	长度（Byte）	描述
Object ID	2	米家定义ID
Object Data Length	1	数据长度
Object Data	若干最大7个字节	具体的数据

按照 MiBeacon 的定义，光照传感器的 Object ID 是 0x1007，数据长度 3 个字节，数值范围是 0~120000 之间。

我将代码贴在下面，供你参考。

复制代码

```
#File: ble_lightsensor.py
import bluetooth
import struct
import time
from ble_advertising import advertising_payload

from micropython import const

_IRQ_CENTRAL_CONNECT = const(1)
_IRQ_CENTRAL_DISCONNECT = const(2)
_IRQ_GATTS_INDICATE_DONE = const(20)

_FLAG_READ = const(0x0002)
_FLAG_NOTIFY = const(0x0010)

_ADV_SERVICE_DATA_UUID = 0xFE95
_SERVICE_UUID_ENV_SENSE = 0x181A
_CHAR_UUID_AMBIENT_LIGHT = 'FEC66B35-937E-4938-9F8D-6E44BBD533EE'

# Service environmental sensing
_ENV_SENSE_UUID = bluetooth.UUID(_SERVICE_UUID_ENV_SENSE)
# Characteristic ambient light density
_AMBIENT_LIGHT_CHAR = (
    bluetooth.UUID(_CHAR_UUID_AMBIENT_LIGHT),
    _FLAG_READ | _FLAG_NOTIFY ,
)
```

```
_ENV_SENSE_SERVICE = (
    _ENV_SENSE_UUID,
    (_AMBIENT_LIGHT_CHAR,),
)

# https://specificationrefs.bluetooth.com/assigned-values/Appearance%20Values.
_ADV_APPEARANCE_GENERIC_AMBIENT_LIGHT = const(1344)

class BLELightSensor:
    def __init__(self, ble, name='Nodemcu'):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        ((self._handle,),) = self._ble.gatts_register_services((_ENV_SENSE_SER
        self._connections = set()
        time.sleep_ms(500)
        self._payload = advertising_payload(
            name=name, services=[_ENV_SENSE_UUID], appearance=_ADV_APPEARANCE_
        )
        self._sd_adv = None
        self._advertise()

    def _irq(self, event, data):
        # Track connections so we can send notifications.
        if event == _IRQ_CENTRAL_CONNECT:
            conn_handle, _, _ = data
            self._connections.add(conn_handle)
        elif event == _IRQ_CENTRAL_DISCONNECT:
            conn_handle, _, _ = data
            self._connections.remove(conn_handle)
            # Start advertising again to allow a new connection.
            self._advertise()
        elif event == _IRQ_GATTS_INDICATE_DONE:
            conn_handle, value_handle, status = data

    def set_light(self, light_den, notify=False):
        self._ble.gatts_write(self._handle, struct.pack("!h", int(light_den)))
        self._sd_adv = self.build_mi_sdadv(light_den)
        self._advertise()
        if notify:
            for conn_handle in self._connections:
                if notify:
                    # Notify connected centrals.
                    self._ble.gatts_notify(conn_handle, self._handle)

    def build_mi_sdadv(self, density):

        uuid = 0xFE95
        fc = 0x0010
        pid = 0x0002
        fcnt = 0x01
        mac = self._ble.config('mac')
```

```
        objid = 0x1007
        objlen = 0x03
        objval = density

        service_data = struct.pack("<3HB",uuid,fc,pid,fcnt)+mac+struct.pack("<
        print("Service Data:",service_data)

        return advertising_payload(service_data=service_data)

    def _advertise(self, interval_us=500000):
        self._ble.gap_advertise(interval_us, adv_data=self._payload)
        time.sleep_ms(100)

        print("sd_adv",self._sd_adv)
        if self._sd_adv is not None:
            print("sdddd_adv",self._sd_adv)
            self._ble.gap_advertise(interval_us, adv_data=self._sd_adv)
```

复制代码

```
#File: main.py
from ble_lightsensor import BLELightSensor
from lightsensor import LightSensor
import time
import bluetooth

def main():
    ble = bluetooth.BLE()
    ble.active(True)
    ble_light = BLELightSensor(ble)

    light = LightSensor(36)
    light_density = light.value()
    i = 0

    while True:
        # Write every second, notify every 10 seconds.
        i = (i + 1) % 10
        ble_light.set_light(light_density, notify=i == 0)
        print("Light Lux:", light_density)

        light_density = light.value()
        time.sleep_ms(1000)

if __name__ == "__main__":
    main()
```

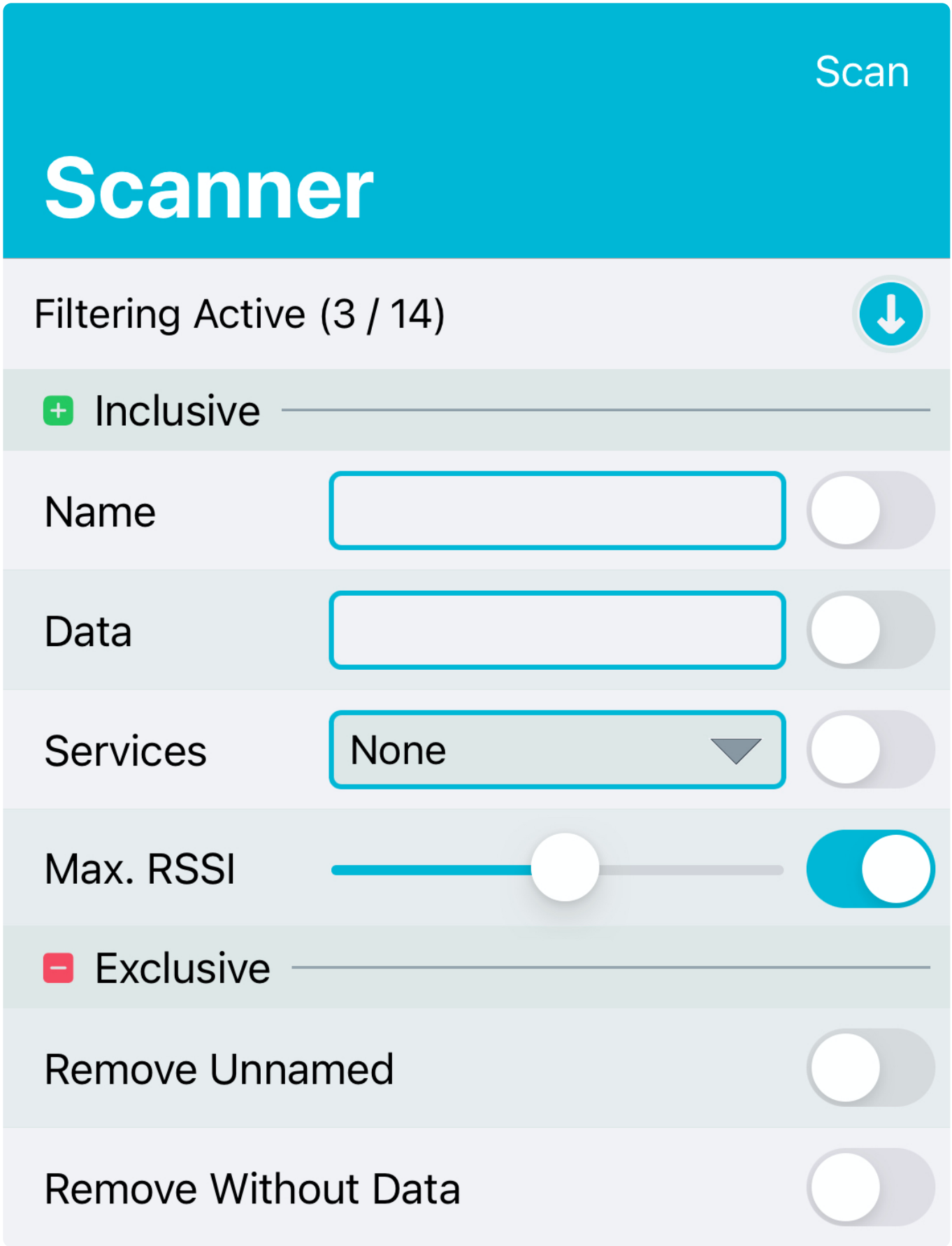
第六步：验证光照传感器

到这里，我们已经完成了光照传感器设备的开发工作。那么怎么验证设备有没有正常工作呢？

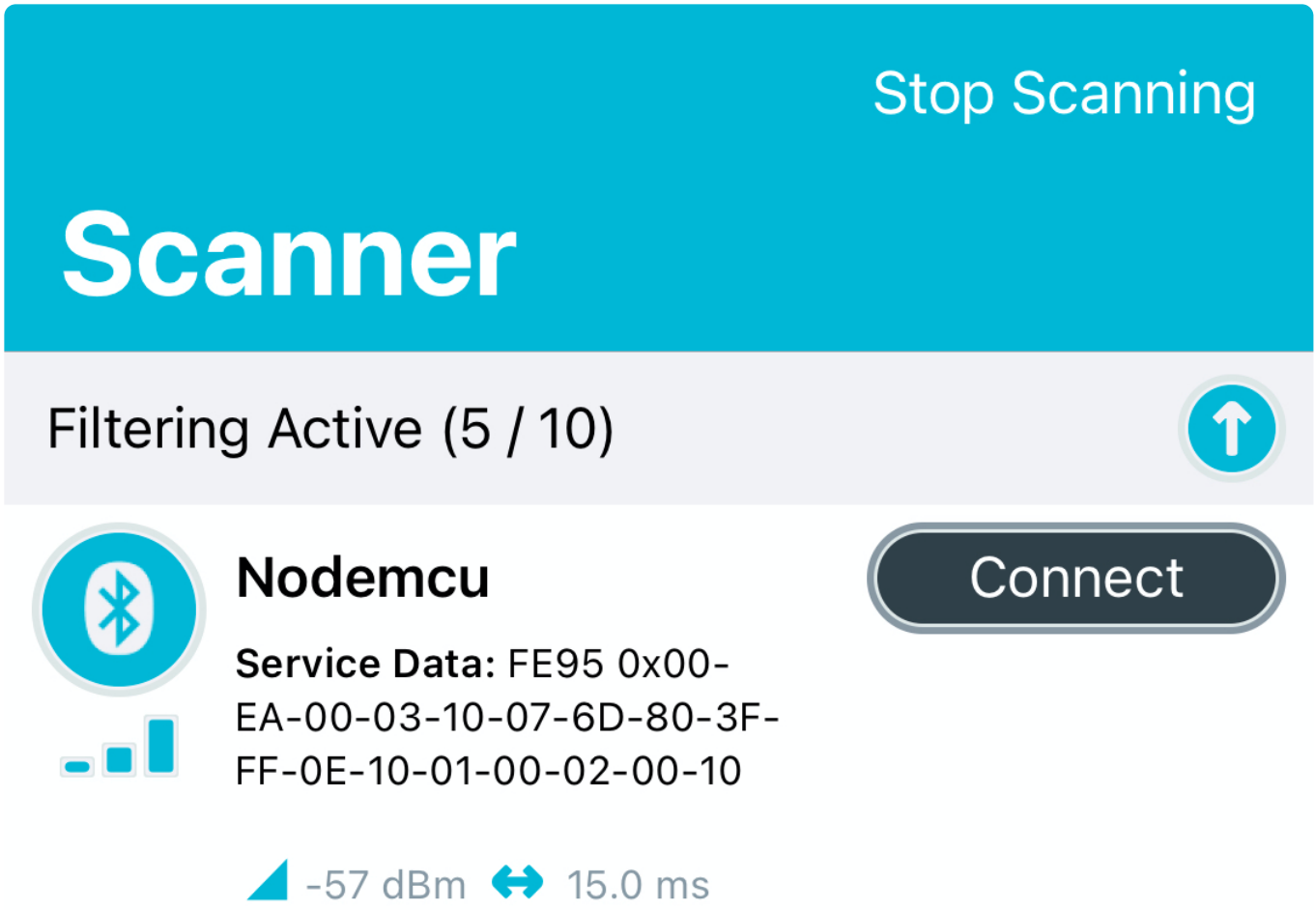
我们可以通过手机上的蓝牙调试软件来扫描周围蓝牙设备，查看设备有没有蓝牙广播包输出，能不能跟手机正常交互。常用的软件有 LightBlue、nRFConnect 和 BLEScanner，选择其中一个就行了。

比如我选择的是 nRF Connect，打开之后，它会自动扫描周围的蓝牙广播包，将发现的设备以列表的形式展示。

如果周围蓝牙设备很多的话，为了方便发现自己的开发板，你可以点击列表上方的“No Filter”，选择将“Max.RSSI”打开。拖动其中的滑竿到合适的值，比如 -50dBm，就可以过滤掉蓝牙信号强度比较弱（一般也是比较远）的设备。



下面是我的手机扫描到的基于 NodeMCU 开发板的蓝牙设备。



其中名称 Nodemcu 下面的就是广播包的具体数据。

到这里，我们就完成了光照传感器设备的开发工作。

小结

总结一下，在这一讲中，我介绍了光照传感器的开发过程，并且补充了低功耗蓝牙技术的相关知识。下面，我们回顾以下重点：

1. 对于无法连接电源线、需要灵活放置甚至经常移动的设备，低功耗蓝牙技术是合适的通信技术选择。
2. MiBeacon 协议的广播包定义是基于 BLE 的 GAP（Generic Access Profile）制定的，主要有广播数据（Advertising Data）和扫描回复数据（Scan Response）两种。其中广播数据中 Service Data 的 UUID 是 0xFE95，扫描回复数据中 Manufacturer Specific Data 的厂家识别码是 0x038F。
3. 在日常的蓝牙设备开发工作中，我们经常需要调试、测试蓝牙功能，这时你可以使用手机上的蓝牙调试软件来验证，比如 LightBlue、nRFConnect 和 BLEScanner 等。

不过，准备好光照传感器设备只是第一步，为了实现光照传感器和智能电灯的联动，我们还需要将光照传感器接入网络。这就需要借助蓝牙网关设备了，在下一讲中，我将基于树莓派讲解网关设备的开发过程。

思考题

最后，给你留一个思考题吧。

在这一讲的开头，我提到蓝牙设备除了广播数据的能力，还可以连接进行交互。在我提供的代码中其实也包含了一个可供连接获取数据的 Service 和 Characteristic，你发现了吗？你知道这些是基于低功耗蓝牙中的什么 Profile 协议吗？

欢迎你在留言区写下自己的答案和我交流一下，也欢迎你将这一讲分享给你的朋友，一起讨论学习。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | 远程控制：怎样打造联网的智能电灯？

下一篇 加餐一 | 这5本关于物联网的好书，值得一读

精选留言 (3)

写留言

Lee
2020-12-22

请问第四步的电路图是用的什么软件呀？看着3D模组做的很好看呀；

LDxy
2020-12-21

在我提供的代码中其实也包含了一个可供连接获取数据的 Service 和 Characteristic，你发现了吗？你知道这些是基于低功耗蓝牙中的什么 Profile 协议吗？

```
        self._ble.gatts_write(self._handle, struct.pack("!h", int(light_den)))
...
```

展开

加油加油

2020-12-21

老师您好，关于开发板这类的硬件知识如何学习比较好？我平时是做云平台的业务，对硬件部分有点陌生